5-1970

A Fortran List Processor (FLIP)

Karl A. Fugal
Utah State University

Follow this and additional works at: https://digitalcommons.usu.edu/etd

Part of the Mathematics Commons, and the Statistics and Probability Commons

Recommended Citation

This Thesis is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Theses and Dissertations by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.
A FORTRAN LIST PROCESSOR (FLIP)

by

Karl A. Fugal

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Applied Statistics

Approved:

UTAH STATE UNIVERSITY
Logan, Utah

1970
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>KNOWN LIST AND STRING PROCESSING LANGUAGES; THEIR CAPABILITIES AND RESTRICTIONS</td>
<td>3</td>
</tr>
<tr>
<td>A FORTRAN LIST PROCESSOR (FLIP)</td>
<td>10</td>
</tr>
<tr>
<td>LITERATURE CITED</td>
<td>18</td>
</tr>
<tr>
<td>APPENDIXES</td>
<td>19</td>
</tr>
<tr>
<td>Appendix A. Subroutine SETUP</td>
<td>20</td>
</tr>
<tr>
<td>Appendix B. Subroutine IAVAIL</td>
<td>25</td>
</tr>
<tr>
<td>Appendix C. Subroutine SLINK</td>
<td>27</td>
</tr>
<tr>
<td>Appendix D. Subroutine LINK</td>
<td>30</td>
</tr>
<tr>
<td>Appendix E. Subroutine STASH</td>
<td>32</td>
</tr>
<tr>
<td>Appendix F. Subroutine GET</td>
<td>36</td>
</tr>
<tr>
<td>Appendix G. Subroutine ERASE</td>
<td>39</td>
</tr>
<tr>
<td>Appendix H. Sample Problems</td>
<td>42</td>
</tr>
<tr>
<td>VITA</td>
<td>61</td>
</tr>
</tbody>
</table>
ABSTRACT

A FORTRAN LIST PROCESSOR (FLIP)

by

Karl A. Fugal

Master of Science

Utah State University, 1970

Major Professor: Wendell L. Pope
Department: Applied Statistics and Computer Science

A series of Basic Assembler Language subroutines were developed and made available to the FORTRAN IV language processor which makes list processing possible in a flexible and easily understood way.

The subroutines will create and maintain list structures in the computer's core storage. The subroutines are sufficiently general to permit FORTRAN programmers to tailor list processing routines to their own individual requirements. List structure sizes are limited only by the amount of core storage available.

(61 pages)
INTRODUCTION

The modern high speed digital computer, in its most general application, can be thought of as a symbol manipulator. However, it is most often used to process numerical data because most widely used programming languages available for the digital computer are designed for numerical calculations for either scientific or business oriented data. When problems arise that require the symbol manipulation capability of the computer, one must transform the problem to operations on numerical data or learn a new programming language designed specifically for symbol manipulation problems.

Several list processing and string processing languages are in existence that are used to program symbol manipulation problems. Most of these existing languages have restrictions and predefined conventions that make them difficult to use by anyone other than a professional programmer. In addition, most of them cannot be used as subroutines to the FORTRAN (FORmula TRANslation) language; that is, they are independent language translators which in turn implies that the programmer must rely entirely on the instruction set afforded by one and only one of these languages.

This thesis contains the documentation and assembly listings for seven subroutines written in Basic Assembler Language for the IBM/360 computer. It is the purpose of
this project to add list processing capabilities to a widely known programming language, namely FORTRAN, in a more flexible and general way than has been done heretofore. The size of the data list is limited only by available core storage. The size of each field within a node is limited to 256 bytes. These subroutines may be used on any IBM S/360 computer that uses the FORTRAN IV language processor.

It is assumed that any potential user of these subroutines has a working knowledge of the FORTRAN language and is familiar with the concept of list processing.
KNOWN LIST AND STRING PROCESSING LANGUAGES:
THEIR CAPABILITIES AND RESTRICTIONS

Many tasks exist which can be performed on a digital computer without knowing specific values of the many variables involved. Getting the computer to perform these tasks can create a communications problem that reaches beyond the capabilities of formal computer language translators (1).

The subroutines developed in this thesis will ease the above mentioned communications problem considerably.

A list may be defined as a group of logically associated items whose sequence, relative to each other, contributes to the meaning of the group. A page taken from a book is an example of a list, where each sentence on that page may be considered an item. Clearly the sequence of this group of sentences is important. List processing is the ability to create, change the sequence of, add to, delete from, and retrieve information from a list or lists. A string may be defined as a variable length sequence of characters and may be considered as one type of list (5). The above example of a group of sentences, or a written page, may be called a string. String processing consists of searching for patterns and transforming them into other patterns, and making insertions and deletions in the string itself.
In order to process or manipulate symbolic data, many list processing and string processing languages have been developed, the oldest of these being the IPL family culminating in IPL-V.

IPL-I (Information Processing Language - I) was a list processing language designed to handle applications involving proving theorems in propositional calculus and playing chess. The first implemented version was IPL-II. This was implemented on the JOHNNIAC computer by the RAND Corporation (5). IPL-III was never implemented because of core storage space problems. IPL-IV was used in the field of artificial intelligence, but was replaced by IPL-V before documentation was finalized and the version implemented.

IPL-V is at a very low language level (almost assembly like) for a list processor. It requires a professional programmer to use it effectively. It has more than 200 primitive subroutines. Probably the most significant contributions made by the IPL family were that they set a groundwork for the design and development of future list processing languages and that they added to the technology of programming in general (5).

L⁶ (Bell Telephone Laboratories Low-Level Linked List Language) was developed in 1965 by Kenneth C. Knowlton (5). It, too, is a list processing language. The internal structure of L⁶ is very different and more efficient than IPL-V. The use of L⁶, its capabilities and restrictions do, however, resemble those of IPL-V.
In 1959, the Artificial Intelligence Group at Massachusetts Institute of Technology (M. I. T.), under the direction of Professor John McCarthy, began work on the LISP programming system designed to facilitate experiments with a proposed system called the Advice Taker, whereby a machine could be instructed to handle declarative as well as imperative sentences and could exhibit "common sense" in carrying out its instructions....The main requirement was a programming system for manipulating sentences so that the Advice Taker system could make deductions.

In the course of its development the LISP system went through several stages of simplification and eventually came to be based on a scheme for representing the partial recursive functions of a class of symbolic expressions (2, p. 405).

LISP is ill suited for anything except general symbol manipulation and list processing. It is meant to be used only by experienced professional programmers. It depends heavily on the use of matching parentheses and is therefore an error-prone language. The LISP language is well adapted to applications that require large amounts of recursion (5).

The first of the string processing languages was COMIT. This system was developed at M. I. T. as a joint project of the Mechanical Translation Group of the Research Laboratory of Electronics and the Computation Center. The system was designed to provide the professional linguist with a computer aid to his research (5). It was intended that nonprofessional programmers be able to write programs in the COMIT language, i.e., the professional linguist himself. COMIT was the first programming system to provide an effective
means of searching for a given string pattern and then performing transformations in that string.

SNOBOL was developed by adding to COMIT, mainly in the areas of string naming and arithmetic capabilities. Work on SNOBOL was started in 1962 at Bell Telephone Laboratories. Later developments and improvements to the language eventually led to the creation of SNOBOL3 and later SNOBOL4.

Other less widely known list processing languages include:

1. TRAC (Text Reckoning and Compiling)
2. TREET
3. CLIP (Cornell List Processor)
4. CORAL (Class Oriented Ring Associative Language)
5. SPRINT
6. LOLITA (Language for the On-Line Investigation and Transformation of Abstractions)

There have been previous attempts to develop a set of primitive subroutines that, when called by a higher level language, provide the capability to do list and/or string processing. Perhaps the most widely known subroutine sets are SLIP and SAC-1. Since this thesis involves the development of another subroutine set that may be embedded in a high level language, i.e., FORTRAN, a more detailed discussion of SLIP and SAC-1 will be presented.

SLIP (Symmetric List Processor) is a descendant of at least four earlier list processors: (a) FLPL by Gelernter;
(b) IPL-V by Newell; (c) Threaded Lists by Perlis; and (d) KLS by Weizenbaum (1).

The fundamental information module with which SLIP deals is a word pair. The first word of the pair is divided into an identification field, a left link field, and a right link field. The second word of the pair is used to contain data (3). A relatively complete set of subroutines and functions are provided by SLIP. This is possible in part by the fixed node structure of SLIP (word pair). The node structure has a distinct disadvantage for many applications in that the node size is fixed and unchangeable, space is required for two link fields even though one field may be sufficient and more than one node is required to store data that are more than one word long. The programmer who uses the SLIP subroutines has to become familiar with several functions and subroutines and in many cases design his application to be compatible with the processor rather than having the advantage of writing a list processing program to fit the application.

SAC-1 (System for Symbolic and Algebraic Calculations - version 1) is a computer independent set of subroutines that are called from a FORTRAN main-line program. SAC-1 uses a relatively small number of simple "primitive subprograms written in an assembly language. The remainder of the SAC-1 list processing system consists of several subprograms written in FORTRAN, the majority of which rely on the
prewritten "primitives". SAC-1 is not an elaborate or extensive list processing system. It provides only the most basic and most essential list processing operations. The user is expected to augment the subprograms of SAC-1 with his own subprograms in order to develop a system that has the capabilities required of it. It should be noted, however, that these user-written subprograms can be written in the FORTRAN language, and thus the use of a lower level language can be avoided.

The node structure defined in SAC-1 is a fixed length group of cells that consist of the type field, the element field, the reference count field, and the successor field. The element field contains the data to be stored in the node (4). The SAC-1 node structure poses the same variable length restriction as does the SLIP node structure. The field lengths of a SAC-1 cell are defined and fixed at the time the "primitive" assembler subprograms are implemented at each installation. The list processing system described by this thesis (FLIP) is very similar in appearance to SAC-1 in that the basic concept of the system is the addition of a small but powerful group of assembly language subroutines to the FORTRAN language. From these "primitive" subroutines a more powerful and more specialized list processing system can be constructed by use of FORTRAN programs and subroutines. SAC-1 has had many powerful FORTRAN subprograms added to it since its initial implementation. Integer arithmetic,
polynomial read and write, and polynomial manipulation routines are some examples.

FLIP is more versatile and flexible than SAC-1 in that the node structure is not fixed. The user may design a node structure consistent with the needs of his application by specifying the number of fields per node and the length (in bytes) of each field.
A FORTRAN LIST PROCESSOR (FLIP)

Description

FLIP is a set of seven assembler language subprograms which may be called by a FORTRAN program. These seven subprograms enable a programmer to design and implement his own list processing language. The subprogram names are SETUP, IAVAIL, LINK, SLINK, GET, STASH, and ERASE. These seven names become reserved words in any program using the subprograms. In this discussion a list will consist of a set of nodes linked together by pointers, each node consisting of a pointer stored in the link field and any number of additional fields. The pointers will be referred to as link variables. The link variables may, at the option of the programmer, point forward or backward. Nodes may be added to the list at either end, thus providing the capability of creating a queued (first in, first out) or stacked (first in, last out) list. Node fields may be used to store additional link variables which will allow the creation of multiple linked lists.

Fields of any length up to 256 bytes may be defined in each node. The programmer has the capability of adding to, deleting from, or changing the sequence of his list at any time. Two or more lists may be combined to form a single list, and a list may be segmented into two or more
other lists. FLIP provides the capability of creating and processing compiler list structures as well as more elementary lists. An attempt was made to hold the number of primitive subroutines to a minimum and make them easy to use by the non professional programmer.

A distinction must be made between commonly used FORTRAN variables, link variables, and field names. FORTRAN variables are: integer, real, subscripted, complex, double precision, etc. Link variables are variables whose values are restricted to addresses and must be of the integer full word type. Field names are used to uniquely identify each field within a node. They are actual FORTRAN variables and as such must be defined before any reference is made to them. Field name variables may be of either the real or integer type.

Methodology

A list of nodes will be constructed in core and a corresponding control table will be developed to carry information needed to access the list. The list will be referred to as the available list. From it the programmer may take and/or return nodes as necessary during the construction of his own list or lists. The available list and control table will be created in an area of core storage reserved by the FORTRAN program. The core storage address of the available list must be available at all times during
the execution of the program. It therefore is stored as a four byte address constant beginning in byte 12 of the communication region in the Disk Operating System supervisor.

SETUP is the name of the subprogram that accepts the reserved core storage from the calling program and creates the available list and control table. SETUP must be invoked once and only once during the execution of the FORTRAN program. It is activated by CALL SETUP (argument list). SETUP creates each node in the available list in the format defined by the argument list and then links the list to form a stack. The calling sequence has the following form: CALL SETUP (vn, d, lfn, 2, fn₁, l₁, fn₂, l₂, ..., fnₖ, lₖ) where vn is the variable name of a subscripted variable occurring in a preceding DIMENSION statement, d is an integer less than or equal to the number of full words in that array, lfn is the link field name the user chooses to use to identify the link field of each node, 2 is the number of bytes in the link field. Each fnᵢ, i = 1, ..., K, is a unique field name of a field in the node, and lᵢ, i = 1, ..., k, is the length, in bytes, of the field named by fnᵢ. The lfn and integer 2 parameters are used only for documentation and to maintain consistency since the link field is always the first two bytes in each node.

The control table is created and stored in the first segment of the array vn. The format of the control table is alp, fn₁, l₁, ..., fnₖ, lₖ where alp is the available
list pointer which is the link to the next available node, $f_{n_1}$, $i = 1, ..., k$, are the field names as discussed above, and $l_i$, $i = 1, ..., k$, are the corresponding field lengths in bytes. Each field name is four bytes in length and each field length is a two byte integer, thus the total length of the control table for a given FORTRAN program can be calculated as $6K + 2$ where $K$ is the number of fields per node and the constant 2 is the number of fields per node and the constant 2 is the number of bytes used for the available list pointer. The SETUP subprogram next creates a series of nodes and links them together to form the available list. The number of nodes that will be created is dependent upon the amount of core storage remaining in the array vn, and may be determined by the formula:

$$\frac{4d - 2 - 6K}{k} = 2 + \sum_{i=1}^{\ell} l_i$$

All addresses are relative to the first byte of the control table which is stored as an address constant in the subroutine SETUP and is subsequently referred to by other primitive subroutines. Actual addresses are composed of the table address as a base and the two bytes relative address. This addressing method allows any location within 65,536 bytes of the beginning of the array vn to be accessed and requires only two bytes to store all address pointers.
As a result of activating the subroutine SETUP, an address constant is stored in a readily available location, a table is created that fully describes each node as defined by the calling program, and a list of nodes is made available for use by the calling program. The subroutine SETUP is 188 bytes in length.

A programmer may create his own list by obtaining nodes from the available list and linking them together. A link variable is returned as the value of the integer valued function IAVAIL. IAVAIL may be activated by a reference such as NA = IAVAIL(X). X is a dummy argument not used by the subprogram. The link variable returned is taken from the first two bytes of the control table. That link variable is then replaced by the link variable of the next node in the available list. This cycle is repeated each time IAVAIL is invoked. When the nodes in the available list have been exhausted, the value of IAVAIL becomes zero. The subprogram IAVAIL requires 40 bytes of core storage.

After a new node is obtained, it can be linked to another node or another node may be linked to it or both links may be made. The SLINK subroutine subprogram is used to perform this linkage. This subroutine stores the two-byte link variable of one node in the link field of another node. SLINK is activated by CALL SLINK (lv, n) where lv is a link variable that is stored in the node pointed to by n (n is thus a link variable also). If reverse linkage is
desired, the arguments lv and n would have to be written in the reverse order, i.e., CALL SLINK (n, lv). In the event a programmer is creating a double linked list, he must use SLINK for one way linkage and the subprogram STASH for the second linkage. STASH will be discussed later. By the repeated use of IAVAIL and SLINK a programmer can thus create a list consisting of as many nodes as is required. The subprogram SLINK requires 70 bytes of core storage.

LINK is an integer valued function subprogram activated by a call such as ID = LINK (lv). Its purpose is to retrieve the value of the link field of the node pointed to by the link variable lv. The contents of the first two bytes of the node referenced by lv are passed back as the returned value. In this manner the list variable of the next sequential node is obtained. If the link variable of the node in sequence beyond the next node is desired, ID = LINK (LINK (lv)) may be invoked. This nesting is valid for as many levels as the IBM S/360 FORTRAN compiler permits. The subroutine LINK requires 48 bytes of core storage.

Data in any machine readable form may be stored in the fields of each node. The data are stored by field through the activation of the subroutine subprogram STASH, i.e., CALL STASH (lv, fn\_1, v\_1, fn\_2, v\_2, ..., fn\_k, v\_k) where lv is a link variable pointing to the receiving node, fn\_i, i = 1, ..., k, are the field names of the receiving fields, and v\_i, i = 1, ..., k, are the values to be stored. "v" may
be any valid FORTRAN variable, subscripted or unsubscripted. Data are transferred beginning with the first byte in \( v \). The number of bytes transferred is equal to the value of the length field of \( fn \) found in the control table. The subroutine \textsc{stash} requires 148 bytes of core storage.

Data stored in a field by the \textsc{stash} subroutine may be retrieved by the \textsc{get} subroutine subprogram. It is activated by \textsc{call get (lv, fn}_1, v_1, fn_2, v_2, \ldots, fn_k, v_k) \text{ where lv is a link variable pointing to the node containing the desired information, fn}_i, i = 1, \ldots, k, \text{ are the field names of the fields containing the desired information, and } v_i, i = 1, \ldots, k, \text{ are the variables capable of receiving the retrieved information. "v" can be a subscripted or unsubscripted variable. Data are transferred to } v \text{ for a length equal to the value of the length field of } fn \text{ as stored in the control table. If the length of } v \text{ exceeds the length of } fn, \text{ information is stored left justified in } v. \text{ All data transferred by the \textsc{get} and \textsc{stash} subprograms are processed without regard to mode. It is therefore imperative that the user assure himself that real variables are used to receive real values and integer variables are used to receive integer values or use some other means to preserve mode compatibility. The subroutine \textsc{get} requires 132 bytes of core storage.}

Nodes that are no longer of any value to a particular list may be returned to the available list by means of the subroutine subprogram \textsc{erase}. This subroutine maintains a
current list of available space. By using ERASE, the problem programmer prevents the accumulation of non active core storage and thus precludes the necessity of the commonly known function called garbage collection. Since the number of nodes available at any given time is limited, it may be important to return any nodes as soon as their purpose has been served. ERASE is activated by CALL ERASE (lv). It returns the node pointed to by the list variable lv to the top of the available list. This node then becomes the next available node and the former first node of the available list linked to it. The contents of returned nodes are not changed, with the exception of the link fields in each node. The subroutine ERASE requires 60 bytes of core storage.


Appendix A

Subroutine SETUP

The following is a source statement listing of
Subroutine SETUP.
SUBROUTINE SETUP

SETUP START 0
USING *,R15
B *+8

ADCON DS F
STM 14,12,12(13)

R1 EQU 1
PARAMETER LIST POINTER
R4 EQU 4
INCREMENT THROUGH TABLE
R5 EQU 5
NO. OF BYTES RESERVED
R6 EQU 6
NO. OF BYTES IN TABLE
R7 EQU 7
WORK
R8 EQU 8
NO. OF BYTES PER NODE
R9 EQU 9
WORK
R10 EQU 10
WORK
R11 EQU 11
WORK
R14 EQU 14
RETURN
R15 EQU 15
BASE

CALL SETUP(RESBLK,100,LNK,2,F1,f,F2,8,F3,2,F4,1)
RESBLK IS AREA-DIMENSION RESBLK(100)-RESERVED FOR LIST.
REMAINING PARAMETERS ARE FIELD NAMES AND LENGTHS FOR EACH NODE

LR R10,R1
R10 IS NOW PARAMETER LIST POINTER

MVC ADCON,0(R10)
STORE ADDRESS OF CORE AREA IN

L R4,0(R10)
ADCON FULL WORD
TABLE AND LIST WILL BE
ADDRESSED

RELATIVE TO THIS LOCATION
GET NEXT PARAMETER-AREA
SIZE

PUT SIZE OF DIMENSION IN R5
MULTIPLY BY 4 TO GET SIZE
IN BYTES

BUILD TABLE IN BEGINNING
OF CORE AREA AS

FOLLOW-LIST VAR. FOR
AVAIL LIST-2 BYTES

FIELD 1-4 BYTES
LENGTH OF FIELD
1-2 BYTES

ETC.

R6 ACCUMULATES TABLE LEN.
R4 INCREMENTS THROUGH TBL.

GET NEXT PARAMETER-NAME
OF FIELD-BYPASS

LNK FIELD AND LENGTH

SINCE THEY ARE KNOWN

PUT FIELD NAME IN TABLE
GET NEXT PARAMETER-FIELD
LENGTH

ACCUMULATE NODE LENGTH
PUT LENGTH IN TABLE
LA R6,6(R6)
LA R4,6(R4)
TM 4(R10),X'80' IS THIS LAST PARAMETER
BZ STORNXT NO
L  R7,RSBLKST SET R7 TO BEGINNING OF
     TABLE-R7 WILL
     CONTAIN BASE ADDRESS R4
     WILL CONTAIN
     ACTUAL ADDRESS-DIFFERENCE
     WILL BE
     DISPLACEMENT ADDRESS TO
     BE STORED IN LINK
     FIELD OF EACH NODE
SR  R9,R7 DISPLACEMENT OF AVAIL
     LIST-FIRST ENTRY IN
STH R9,0(R7) TABLE
SR  R5,R6 R5 NOW HAS CORE LEFT FOR
     LIST
     CREATE AVAIL LIST
     
LR  R7,R5 DIVIDE R5 BY R8 TO
SR  R6,R6 DETERMINE HOW MANY
DR  R6,R8 NODES WILL FIT INTO
     REMAINING AREA STORE
     THAT NO. IN R7
     R4 NOW HAS ACTUAL ADDRESS
LNKNXT AR R9,R8

* OF CURRENT NODE

R9 NOW HAS RELATIVE

ADDRESS OF NEXT NODE

STH R9,RSBLKST

FOR ALIGNMENT

MVC 0(2,R4),RSBLKST

AR R4,R8

BCT R7,LNKNXT

SR R4,R8

SET LNK FIELD IN LAST

NODE TO ZEROES

*

STH R7,RSBLKST

FOR ALIGNMENT

MVC 0(2,R4),RSBLKST

LM 2,12,28(13)

MVI 12(13),X'FF'

BR R14

RETURN

RSBLKST DS F

REG SAVE AREA

END
Appendix B

Subroutine IAVAIL.

The following is a source statement listing of Subroutine IAVAIL.
SUBROUTINE IAVAIL

IAVAIL START 0
USING *,R15
STM 14,12,12(13) STORE REGS FROM CALLING

* R0 EQU 0 RESULT
R1 EQU 1 PARAMETER LIST
R4 EQU 4 TABLE ADDRESS-BASE
R5 EQU 5 DISPLACEMENT
R14 EQU 14 RETURN
R15 EQU 15 BASE

* IA-IAVAIL(X)
* IA IS WHERE RESULT IS STORED
* X IS DUMMY ARGUMENT

L R1,=V(SETUP)
L R4,0(R1) BASE IN R4
SR R5,R5
LH R5,0(R4) LINK OF NEXT NODE FROM AVAIL LIST

LR R0,R5 RESULTANT VALUE
AR R5,R4 BASE + DISPLACEMENT
MVC 0(2,R4),0(R5) GET LINK OF NEXT NODE
LM 2,12,29(13) AND PLACE IN TABLE
MVI 12(13),X'FF'
BR R14
END
Appendix C

Subroutine SLINK

The following is a source statement listing of Subroutine SLINK.
SUBROUTINE SLINK

SLINK  START 0
USING *,R15
STM 14,12,12(13)
R1 EQU 1
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R14 EQU 14
R15 EQU 15

* CALL SLINK(IA,NODE)
* IA IS LIST VARIABLE TO BE STORED IN LINK FIELD OF NODE
* NODE IS LIST VARIABLE OF NODE

LR R7,R1
L R6,0(R1) ADDRESS OF IA IN R6
L R5,4(R1) ADDRESS OF NODE IN R5
L R1,=V(SETUP)

LOOP L R4,0(R1) TABLE BASE IN R4
L R5,0(R5)
AR R4,R5
MVC 0(2,R4),2(R6) PUT VALUE OF IA IN LINK
TM 4(R7),X'80' FIELD
BO RET LAST SET OF PARAMETERS
LA R7,8(R7)
L R6,0(R7)
L  R5,4(R7)
B  LOOP
RET  LM  2,12,28(13)
MVI  12(13),X'FF'
BR  R14
END
Appendix D

Subroutine LINK

The following is a source statement listing of Subroutine LINK.
SUBROUTINE LINK

START 0
USING *,R15
STM 14,12,12(13) STORE REGS FROM CALLING PROGRAM

R0 EQU 0 RETURN RESULT
R1 EQU 1 PARAMETER LIST POINTER
R4 EQU 4 ADDRESS OF TABLE
R5 EQU 5 VALUE OF ARGUMENT
R14 EQU 14 RETURN
R15 EQU 15 BASE

* IA=LINK(LINK(LINK(LISTVR)))
* IA IS WHERE RESULTANT LIST ADDRESS IS PLACED--BASE
* DISPLACEMENT FORM
* LISTVR IS NAME OF LIST VARIABLE--DISPLACEMENT FORM-

L R5,0(R1) ADDRESS OF LIST VARIABLE
L R1,V(SETUP)
L R4,0(R1) ADDRESS OF TABLE--BASE-
A R4,0(R5) BASE + DISPLACEMENT
MVC FWRD(2),(R4) ALIGNMENT
LH R0,FWRD
LM 2,12,28(13)
MVI 12(13),X'FF'
BR R14
FWRD DS F
END
Appendix E

Subroutine STASH

The following is a source statement listing of Subroutine STASH.
SUBROUTINE STASH

STASH  START 0
USING *,R15

STM 14,12,12(13)  STORE_REGS_FROM_CALLING

R1  EQU 1  PARAMETER_LIST_POINTER
R4  EQU 4  ADDRESS_OF_TABLE
R5  EQU 5  ADDRESS_OF_LISTV
R6  EQU 6  ADDRESS_OF_NODE_FIELD
R7  EQU 7  LENGTH_OF_NODEFLD_TAKEN
R8  EQU 8  ACCUMULATE_DISPLACEMENT
R9  EQU 9  ADDRESS_OF_INFO_FIELD
R10 EQU 10 SAVE_PARM_LIST_POINTER
R14 EQU 14 RETURN
R15 EQU 15 BASE

CALL STASH(LISTV,NODEFLD,INFO,NODEFLD2,INFO2,-----)

* LISTV IS NAME OF LIST VARIABLE THAT CONTAINS THE LIST
* ADDRESS OF THE NODE.
* NODEFLD IS THE FIELD NAME IN THE NODE.
* INFO IS THE VARIABLE THAT CONTAINS THE INFO TO BE
* STORED.

L R5,0(R1)  LOAD_PARAMETERS
L R5,0(R5)
L R9,8(R1)
L R10,8(R1)
L R1,=V(SETUP) GET ADDRESS OF TABLE FROM SETUP

L R4,0(R1) AND PUT IN R4
LR R8,R4 BASE IN R8
LA R4,2(R4) BYPASS LINK FIELD IN TABLE
LA R8,2(R8) TABLE
STM R4,R5,SV45
ST R8,SV8

NEXT CLC 0(4,R6),0(R4) FIND FIELD NAME IN
BE FOUND TABLE
AH R8,4(R4) ADD FIELD LENGTH FROM
LA R4,6(R4) TABLE TO R8
B NEXT

EXMVC MVC 0(0,R5),0(R9) STORE
FOUND LH R7,4(R4) FIELD LENGTH IN R7
AR R5,R8 BASE + DISPLACEMENT OF FIELD

BCTR R7,0 SUBTRACT ONE FROM R7
EX R7,EXMVC FOR EX COMMAND
TM 0(R10),X'80'
BO FINIS
L R6,8(R10) NEXT PARAMETERS
L R9,8(R10)
LA R10,8(R10)
LM R4,R5,SV45
L R8,SV8
B NEXT
FINIS EQU *

LM 2,12,28(13) RESTORE REGS FROM CALLING

MVI 12(13),X'FF' PROG

BR R14

SV45 DS 2F

SV8 DS F

END
Appendix F

Subroutine GET

The following is a source statement listing of Subroutine GET.
SUBROUTINE GET

GET START 0

USING *,R15

STM 14,12,12(13)

R0 EQU 0 RESULT
R1 EQU 1 PARAMETER POINTER
R4 EQU 4 BASE
R5 EQU 5 NODE VALUE - DISPLACEMENT
R6 EQU 6 FIELD NAME OF NODE
R7 EQU 7 DATA LENGTH
R8 EQU 8 DISPLACEMENT WITHIN NODE
R9 EQU 9 POINT TO VINFO
R10 EQU 10 SAVE PARM LIST POINTER
R15 EQU 15

* CALL GET(NODE,FIELD1,VINFO1,FIELD2,VINFO2,---------)
* NODE IS LIST VARIABLE FOR NODE OF INTEREST
* FIELD1 IS FIELD OR CELL NAME WHERE INFORMATION IS
* STORED
* VINFO IS SYMBOLIC NAME OF CORE LOCATION WHERE THE
* INFORMATION WILL BE PLACED

L R5,0(R1) PUT NODE VALUE IN R5
L R5,0(R5)
L R6,4(R1) R6 POINTS TO FIELD VALUE
L R9,8(R1) POINT TO VINFO
LA R10,8(R1)
L R1,=V(SETUP)
L R4,0(R1) ADDRESS OF TABLE
AR  R5,R4                BASE + DISPLACEMENT
LA  R4,2(R4)             BYPASS LINK FIELD
LA  R5,2(R5)             LENGTH OF LINK FIELD
STM R4,R5,SV45

NEXT CLC 0(4,R6),0(R4)   FIELD IN TABLE
BE  FOUND
AH  R5,4(R4)
LA  R4,6(R4)
B  NEXT

EXMVC MVC 0(0,R9),0(R5)

FOUND LH R7,4(R4)        LENGTH OF DATA FIELD
BCTR R7,RO              SUBTRACT ONE FOR EXEC
EX  R7,EXMVC            COMMAND
TM  O(R10),X'80'        NEXT FIELD VALUE
BO  FINIS
LM  R4,R5,SV45

LA  R6,4(R10)           NEXT FIELD VALUE
LA  R9,8(R10)           NEXT VINFO
LA  R10,8(R10)
B  NEXT

FINIS EQU *

LM  2,12,28(13)         RESTORE REGS FROM CALLING
MVI 12(13),X'FF'        PROGRAM
BR  14

SV45 DS 2F

END
Appendix G

Subroutine ERASE

The following is a source statement listing of Subroutine ERASE.
* SUBROUTINE ERASE

ERASE  START 0
USING *,R15
STM 14,12,12(13)
R1 EQU 1
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R14 EQU 14
R15 EQU 15

* CALL ERASE(IA)
* RETURN TO AVAILABLE LIST THE NODE REFERENCED BY THE
* LIST VARIABLE IA
* ALSO SET VALUE OF IA TO ZERO SO THAT IA CAN NO LONGER
* BE USED WITHOUT BEING RESTORED

L R6,9(R1) ADDRESS OF IA
L R1,=V(SETUP)
L R4,0(R1) TABLE BASE IN R4
LH R5,0(R4)
LR R8,R6
AR R5,R4 PUT TABLE IN LINK
*
L R6,0(R6) BEING RETURNED
LR R7,R6 SAVE BASE VALUE OF IA
AR R6,R4
MVC 0(2,R6),0(R4)

STH R7,0(R4)  PUT IA IN TABLE LINK

SR R1,R1

ST R1,0(R8)  ZERO OUT IA

LM 2,12,28(13)

MVI 12(13),X'FF'

BR R14

END
Appendix H

Sample Problems

Three sample programs have been written to demonstrate the use of the seven FLIP subroutines. The first is the multiplication of two polynomials. The input is one header card followed by one or more coefficient and exponent cards for each polynomial. The header card contains the variable and the number of terms in the polynomial. The format for the header card is:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2</td>
<td>Number of terms in polynomial</td>
</tr>
<tr>
<td>3</td>
<td>Variable name</td>
</tr>
</tbody>
</table>

The format of the data cards is:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-10</td>
<td>Coefficient term</td>
</tr>
<tr>
<td>11-12</td>
<td>Exponent term</td>
</tr>
</tbody>
</table>

The second sample is the division of two polynomials. The input data format is the same as in the above sample with the dividend taken to be the first polynomial and the divisor taken to be the second.

The third sample will read in and create a list of names, social security numbers, birth years, high school codes, and the sex of a given group of people. A sort
will then be done on social security number and the list printed out in sequence by social security number. It should be noted that the only data movement will be that of the social security numbers and their corresponding list variables.

The format for the data is:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-22</td>
<td>Name</td>
</tr>
<tr>
<td>27-28</td>
<td>Year of birth</td>
</tr>
<tr>
<td>30</td>
<td>Sex</td>
</tr>
<tr>
<td>38-40</td>
<td>High school code</td>
</tr>
<tr>
<td>43-51</td>
<td>Social Security Number</td>
</tr>
</tbody>
</table>
C POLYNOMIAL MULTIPLY
DIMENSION POLYNOMIAL MULTIPLY
COEF=1.0
EXP=2.0
CALL SETUP(RESBLK,500,LNK,2,COEF,4,EXP,4)
LV1=IAVAIL(X)
LV2=IAVAIL(X)
LVP=IAVAIL(X)
L1=LV1
N=0
LP=LVP
PRINT 4
FORMAT(8X,'MULTIPLICAND')
READ 1,N1,V1
FORMAT(I2,A1)
LOOP=1
READ 2,COF,IEX
FORMAT(F10.1,I2)
CALL STASH(L1,COEF,COF,EXP,IEX)
PRINT 8,COF,V1,IEX
IF(LOOP .EQ. N1) GO TO 5
LOOP=LOOP+1
L=IAVAIL(X)
CALL SLINK(L,L1)
L1=L
GO TO 6
5 L1=LV2
N=N+1
IF(N .EQ. 2) GO TO 11
PRINT 12
FORMAT(8X,'MULTIPLIER')
GO TO 7
11 II=2*N1-2
DO 10 I=1,II
L=IAVAIL(X)
CALL SLINK(L,LP)
10 LP=L
CALL POLYMT (LV1, LV2, LVP, N1, COEF, EXP)
PRINT 3
FORMAT(8X,'PRODUCT')
LOOP=1
9 CALL GET(LVP,COEF,C1,EXP,IX)
IF(C1 .EQ. 0.0) GO TO 13
PRINT 8,C1,V1,IX
8 FORMAT(1X,F10.2,A1,'EXP',I2)
13 IF(LOOP .EQ. 2*N1-1) STOP
LOOP=LOOP+1
LVP=LINK(LVP)
GO TO 9
END.
SUBROUTINE POLYMT (LV1, LV2, LVP, NTERMS, COEF, EXP)
C MULTIPLY LV1 BY LV2 RESULT IN LVP.
C ALL CELLS IN LV1 and LV2 MUST BE INITIALIZED.
C IF A TERM IS MISSING, COEF MUST BE SET TO ZERO.
C NTERMS IS NUMBER OF TERMS IN POLYNOMIAL
C COEF IS FIELD NAME OF COEFFICIENT FIELD
C EXP IS FIELD NAME OF EXPONENT FIELD
NPERMS = 2 * NTERMS - 1
LVWRK = IAVAIL(X)
LVW2 = LVWRK
LVW3 = LVWRK
LV = LVWRK
IN = NTERMS - 1
DO 1 I = 1, NPERMS
    N = IAVAIL(X)
    CALL SLINK(N, LV)
    LV = N
1 CONTINUE
LVV2 = LV2
IOUTER = 1
LVP1 = LVP
LOOP = 1
ZERO = 0.0
IEXP = 2 * NTERMS - 2
6 CALL STASH(LVP1, COEF, ZERO, EXP, IEXP)
   IF (LOOP .EQ. NPERMS) GO TO 5
   LOOP = LOOP + 1
   LVP1 = LINK(LVP1)
   IEXP = IEXP - 1
   GO TO 6
5 LVW1 = LVWRK
   LVV1 = LV1
   CALL GET(LVV2, COEF, C2)
   DO 9 I = 1, NPERMS
      CALL STASH(LVV1, COEF, ZERO)
9   LVW1 = LINK(LVV1)
   LVV1 = LVWRK
   INNER = 1
2 CALL GET(LVV1, COEF, C1)
   C2 = C1 * C2
   CALL STASH(LVW2, COEF, C3)
   IF (INNER .EQ. NTERMS) GO TO 3
   LVV1 = LINK(LVV1)
   LVW2 = LINK(LVW2)
   INNER = INNER + 1
   GO TO 2
3 CALL CELLAD (LVWRK, LVP, NTERMS, COEF, EXP)
   IF (OUTER .EQ. NTERMS) GO TO 7
   IOUTER = IOUTER + 1
   LVV2 = LINK(LVV2)
LVW3 = LINK(LVW3)
LVW2 = LVW3
GO TO 5
7
DO 8 I = 1, IN
LVWL = LINK(LVWRK)
CALL ERASE(LVWRK)
8
LVWRK = LVWL
CALL ERASE(LVWL)
RETURN
END
SUBROUTINE CELLAD (LV1, LV2, NTERMS, COEF)
LVV1 = LV1
LVV2 = LV2
I = 1
1
CALL GET(LVV1, COEF, C1)
CALL GET(LVV2, COEF, C2)
C2 = C2 + C1
CALL STASH(LVV2, COEF, C2)
IF (I .EQ. 2**NTERMS - 1) RETURN
I = I + 1
LVV1 = LINK(LVV1)
LVV2 = LINK(LVV2)
GO TO 1
END
### INPUT

<table>
<thead>
<tr>
<th>5Z</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3.0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5Z</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3.0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>10.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MULTIPLICAND
1.30ZEXP 4
2.00ZEXP 3
3.00ZEXP 2
1.20ZEXP 1
0.0 ZEXP 0

MULTIPLIER
5.00ZEXP 4
0.0 ZEXP 0
4.00ZEXP 2
3.00ZEXP 1
10.00ZEXP 0

PRODUCT
6.50ZEXP 8
10.00ZEXP 7
20.20ZEXP 6
17.90ZEXP 5
31.00ZEXP 4
33.80ZEXP 3
33.60ZEXP 2
12.00ZEXP 1
PROGRAM

C POLYNOMIAL DIVIDE
DIMENSION RESBLK(500)
COEF=1.0
EXP=2.0
CALL SETUP(RESBLK,500,LNK,2,COEF,4,EXP,4)
LDVD=IAVAIL(X)
LDVR=IAVAIL(X)
LQ=IAVAIL(X)
LR=IAVAIL(X)
L1=LDVD
NSW=0
PRINT 8
8 FORMAT(1X'DIVIDEND')
10 READ 1,N1,V
1 FORMAT(I2,A1)
LOOP=1
6 READ 2,COF,IEX
2 FORMAT(F10.1,I2)
CALL STASH(L1,EXP,IEX,COEF,COF)
PRINT 17,COF,V,IEX
IF(LOOP .EQ. N1) GO TO 5
LOOP=LOOP+1
L=IAVAIL(X)
CALL SLINK(L,L1)
L1=L
GO TO 6
5 IF(NSW .EQ. 1) GO TO 7
L1=LDVR
ND=N1
PRINT 9
9 FORMAT(1X,'DIVISOR')
GO TO 10
7 NR=N1
L1=LQ
LOOP=1
COF=0.0
IEX=0
12 CALL STASH(L1,COEF,COF,EXP,IEX)
IF(LOOP.EQ.ND) GO TO 11
L=IAVAIL(X)
CALL SLINK(L,L1)
L1=L
LOOP=LOOP+1
GO TO 14
PRINT 15
15 FORMAT(1X, 'QUOTIENT')
16 LQ=LINK(LQ)
PRINT 18
18 FORMAT(1X, 'REMAINDER')
NRMO=NR-1
DO 19 I=1,NRMO
CALL GET(LR,COEF,C1,EXP,IX)
IF(C1 .EQ. 0.0) GO TO 19
PRINT 17,C1,V,IX
19 LR=LINK(LR)
STOP
END
SUBROUTINE POLYDV(LDVD, ND, LDVR, NR, LQ, LR, COEF, EXP)
C
DIVIDE LDVD WITH ND TERMS BY LDVR WITH NR TERMS,
C
PUT QUOTIENT WITH ND TERMS IN LQ AND REMAINDER
C
WITH NR TERMS IN LR. FIELD NAMES ARE COEF AND
C
EXP FOR COEFFICIENT AND EXPONENT TERMS IN EACH
C
NODE.
WD=IAVAIL(X)
WL=IAVAIL(X)
I=1
COF=0.0
IEX=0
LDW=WD
LL=WL
LDD=LDVD
1
CALL GET(LDD, COEF, CW, EXP, IW)
CALL STASH(LDW, COEF, CW, EXP, IW)
CALL STASH(LL, COEF, COF, EXP, IEX)
IF(I.EQ.ND) GO TO 2
LDD=LINK(LDD)
L=IAVAIL(X)
CALL SLINK(L, LDW)
LDW=L
L=IAVAIL(X)
CALL SLINK(L, LL)
LL=L
I=I+1
GO TO 1
2
LWQ=LQ
LWR=LR
LDW=WD
LLW=WL
LDVRW=LDVR
5
CALL GET(LDVRW, EXP, IEXDVR)
CALL GET(LDW, EXP, IEXDVD)
IF(IEXDVR.GT. IEXDVD) GO TO 3
CALL GET(LDVRW, COEF, CDVR)
CALL GET(LDW, COEF, CDVD)
CQ=CDVD/CDVR
IEXQ=IEXDVD-IEXDVR
LDW=LINK(LDW)
CALL STASH(LWQ, COEF, CQ, EXP, IEXQ)
LSLDW-LDW
LDRN=LINK(LDVR)
NRMO=NR-1
DO 4 I=1, NRMO
CALL GET(LDRN, COEF, CDVRN)
CALL GET(LSLDW, COEF, C1)
CW=C1-CQ*CDVRN
CALL STASH(LSLDW, COEF, CW)
LDRN=LINK(LSDLW)
LSLDW=LINK(LSDLW)
LWQ=LINK(LWQ)
GO TO 5

NRMO=NR-1
DO 6 I=1,NRMO
   CALL GET(LDW,COEF,C1,EXP,IEX)
   CALL STASH(LWR,COEF,C1,EXP,IEX)
   LWR=LINK(LWR)
6
LDW=LINK(LDW)
RETURN
END
SAMPLE INPUT

6X
2.0 5
1.0 4
-5.0 3
9.0 2
12.0 1
2.0 0

3X
1.0 2
2.0 1
-3.0 0
<table>
<thead>
<tr>
<th>DIVIDEND</th>
<th>DIVISOR</th>
<th>QUOTIENT</th>
<th>REMAINDER</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0XEXP 5</td>
<td>1.0XEXP 2</td>
<td>2.0XEXP 3</td>
<td>61.0XEXP 1</td>
</tr>
<tr>
<td>1.0XEXP 4</td>
<td>2.0XEXP 1</td>
<td>-3.0XEXP 2</td>
<td>-40.0XEXP 0</td>
</tr>
<tr>
<td>-5.0XEXP 3</td>
<td>-3.0XEXP 0</td>
<td>7.0XEXP 1</td>
<td></td>
</tr>
<tr>
<td>9.0XEXP 2</td>
<td></td>
<td>-14.0XEXP 0</td>
<td></td>
</tr>
<tr>
<td>12.0XEXP 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0XEXP 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
C

SORT

DIMENSION RESBLK(500),LK(99),NAM(6)

SS=1.0
NAME=2
YBR=3.0
SEX=4.0
HSCL=5.0

CALL SETUP(RESBLK,500,LNK,2,SS,4,NAME,22,YBR,
12,SEX,1,HSCL,4)

DO 5 I=1,1000
READ(5,1,END=2) NAM,IYBR,ISEX,ISCL,ISS

1 FORMAT(5A4,A2,3X,A2,1X,A1,7X,A3,2X,I9)

LK(I)=AVAIL(X)

CALL STASH(LK(I),SS,ISS,NAME,NAM,YBR,IYBR,SEX,
1ISEX,HSCL,ISCL)

5 CONTINUE

2 I=I-1
M=I
N=M/2

3 I=1
J=I+N

7 CALL GET(LK(I),SS,ISS)
CALL GET(LK(J),SS,JSS)
IF(ISS .GT. JSS) GO TO 6

4 IF(J .EQ. M) GO TO 9
J=J+1
I=I+1
GO TO 7

6 LSV=LK(I)
LK(I)=LK(J)
LK(J)=LSV
GO TO 4

9 IF(N .EQ. 1) GO TO 11
N=N-1
GO TO 3

11 WRITE(6,14)

14 FORMAT(1X,'NUMBER',5X,'YR',1X,'SEX',1X,'H.S.',/) 

DO 12 I=1,M
CALL GET(LK(I),SS,ISS,NAME,NAM,YBR,IYBR,SEX,
1ISEX,HSCL,ISCL)

12 WRITE(6,13) ISS,NAM,IYBR,ISEX,ISCL

13 FORMAT(1X,I9,2X,5A4,A2,2X,A2,2X,A1,2X,A3)
STOP

END
<table>
<thead>
<tr>
<th>Name</th>
<th>Gender</th>
<th>Age</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>KARIMI AHMAD</td>
<td>M</td>
<td>48</td>
<td>111</td>
</tr>
<tr>
<td>SAADAT MOHAMAD H</td>
<td>M</td>
<td>50</td>
<td>555</td>
</tr>
<tr>
<td>SMITH LINDA DIANE</td>
<td>F</td>
<td>48</td>
<td>127</td>
</tr>
<tr>
<td>HUNLEY MICHAEL W</td>
<td>M</td>
<td>47</td>
<td>528</td>
</tr>
<tr>
<td>BROWN LAWRENCE GUY</td>
<td>M</td>
<td>39</td>
<td>539</td>
</tr>
<tr>
<td>HARVEILL WILLIAM DEAN</td>
<td>M</td>
<td>49</td>
<td>520</td>
</tr>
<tr>
<td>ANDRE RAMONA LOUISE E</td>
<td>M</td>
<td>34</td>
<td>549</td>
</tr>
<tr>
<td>BARFUS BRENT WAYNE</td>
<td>M</td>
<td>42</td>
<td>529</td>
</tr>
<tr>
<td>ZEHNPFENNING BRENDA K</td>
<td>F</td>
<td>44</td>
<td>528</td>
</tr>
<tr>
<td>PETERSON GLEN LOREN</td>
<td>M</td>
<td>44</td>
<td>529</td>
</tr>
<tr>
<td>SIQUEIROS BRUCE WAYNE</td>
<td>M</td>
<td>49</td>
<td>666</td>
</tr>
<tr>
<td>WALTERS JACK LEROY</td>
<td>M</td>
<td>35</td>
<td>528</td>
</tr>
<tr>
<td>BROWN VERNAL A</td>
<td>M</td>
<td>38</td>
<td>520</td>
</tr>
<tr>
<td>BLACK DAVID F</td>
<td>M</td>
<td>37</td>
<td>527</td>
</tr>
<tr>
<td>COULAM WILLIAM B</td>
<td>M</td>
<td>45</td>
<td>528</td>
</tr>
<tr>
<td>WALKER CLIVE HANSEN</td>
<td>M</td>
<td>35</td>
<td>777</td>
</tr>
<tr>
<td>BARSON AARON V JR</td>
<td>M</td>
<td>48</td>
<td>528</td>
</tr>
<tr>
<td>KELSO THOMAS R</td>
<td>M</td>
<td>50</td>
<td>561</td>
</tr>
<tr>
<td>NASERI MOHSEN</td>
<td>M</td>
<td>50</td>
<td>888</td>
</tr>
<tr>
<td>MANN MICHAEL DOUGLAS</td>
<td>M</td>
<td>50</td>
<td>528</td>
</tr>
<tr>
<td>SMITH DENNIS LYLE</td>
<td>M</td>
<td>47</td>
<td>999</td>
</tr>
<tr>
<td>PRREDEDILOK KITIMA P</td>
<td>F</td>
<td>39</td>
<td>529</td>
</tr>
<tr>
<td>HESTER JAMES CLITTON</td>
<td>M</td>
<td>47</td>
<td>528</td>
</tr>
<tr>
<td>SORENSEN ARTHUR BRYCE</td>
<td>M</td>
<td>29</td>
<td>518</td>
</tr>
<tr>
<td>PLUMMER TIMOTHY N</td>
<td>M</td>
<td>48</td>
<td>505</td>
</tr>
<tr>
<td>WOODS DONA CELESTE</td>
<td>F</td>
<td>49</td>
<td>457</td>
</tr>
<tr>
<td>WEYERTS THOMAS MARTIN</td>
<td>M</td>
<td>48</td>
<td>562</td>
</tr>
<tr>
<td>EDWARDS RONALD DANE</td>
<td>M</td>
<td>47</td>
<td>557</td>
</tr>
<tr>
<td>WORZALA JAMES LYLE</td>
<td>M</td>
<td>42</td>
<td>339</td>
</tr>
<tr>
<td>MARTIN LONNIE JOSEPH</td>
<td>M</td>
<td>47</td>
<td>530</td>
</tr>
<tr>
<td>JARRELL DANIEL CRAIG</td>
<td>M</td>
<td>47</td>
<td>538</td>
</tr>
<tr>
<td>BETHERS BARTON L</td>
<td>M</td>
<td>26</td>
<td>528</td>
</tr>
<tr>
<td>TURNQUIST GARY BRUCE</td>
<td>M</td>
<td>50</td>
<td>573</td>
</tr>
<tr>
<td>LAMOUREUX BLAIN C</td>
<td>M</td>
<td>49</td>
<td>528</td>
</tr>
<tr>
<td>PETERS DONALD CARL</td>
<td>M</td>
<td>47</td>
<td>552</td>
</tr>
<tr>
<td>SIAL MUHAMAD IZBAL</td>
<td>M</td>
<td>48</td>
<td>111</td>
</tr>
<tr>
<td>PEARSON WILLIAM DEAN</td>
<td>M</td>
<td>41</td>
<td>360</td>
</tr>
<tr>
<td>HERREID BARBARA ELLEN</td>
<td>F</td>
<td>48</td>
<td>303</td>
</tr>
<tr>
<td>MURPHY DEE FRANKLIN</td>
<td>M</td>
<td>46</td>
<td>529</td>
</tr>
<tr>
<td>HESS MARGARET MYLER</td>
<td>F</td>
<td>42</td>
<td>519</td>
</tr>
<tr>
<td>REED FRANK E</td>
<td>M</td>
<td>28</td>
<td>532</td>
</tr>
<tr>
<td>TAYLOR STEPHEN COPE</td>
<td>M</td>
<td>48</td>
<td>528</td>
</tr>
<tr>
<td>QUINTANA ROCKY</td>
<td>M</td>
<td>47</td>
<td>528</td>
</tr>
<tr>
<td>HACKLEY LARAINN STUCKI</td>
<td>F</td>
<td>69</td>
<td>518</td>
</tr>
<tr>
<td>CHIDESTER MILTON WAYNE</td>
<td>M</td>
<td>51</td>
<td>528</td>
</tr>
<tr>
<td>BRIGGS BETTE MARLENE</td>
<td>F</td>
<td>48</td>
<td>530</td>
</tr>
<tr>
<td>Name</td>
<td>Age</td>
<td>Sex</td>
<td>Height</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----</td>
<td>-----</td>
<td>--------</td>
</tr>
<tr>
<td>COPE PATRICIA</td>
<td>48</td>
<td>F</td>
<td>529</td>
</tr>
<tr>
<td>TEW RICHARD WAYNE</td>
<td>51</td>
<td>M</td>
<td>461</td>
</tr>
<tr>
<td>GUYNMON MAUGHAN M</td>
<td>47</td>
<td>M</td>
<td>529</td>
</tr>
<tr>
<td>MAJOR GARY LEE</td>
<td>45</td>
<td>M</td>
<td>550</td>
</tr>
<tr>
<td>VAUGHN MICHAEL W</td>
<td>50</td>
<td>M</td>
<td>440</td>
</tr>
</tbody>
</table>
OUTPUT

111  KARIMI AHMAD               48 M
111  SIAL MUHAMAD IZBAL          48 M
303  HERREID BARBARA ELLEN       48 F
339  WORZALA JAMES LYLE          42 M
360  PEARSON WILLIAM DEAN        41 M
440  VAUGHN MICHAEL W            50 M 050
457  WOODS DONA CELESTE          49 F
461  TEW RICHARD WAYNE          51 M
505  PLUMMER TIMOTHY N           48 M
518  SORENSEN ARTHUR BRYCE       29 M 382
518  HACKLEY LARAIN STUCKI       69 F 209
519  HESS MARGARET MYLER         42 F 012
520  BROWN VERNAL A              38 M 620
520  HARVELL WILLIAM DEAN        49 M 570
527  BLACK DAVID F               37 M
528  BETHERS BARTON L            26 M 141
528  WALTERS JACK LEROY          35 M 027
528  ZEHNPFENNING BRENDA K       44 F 145
528  COULAM WILLIAM B            45 M 108
528  QUINTANA ROCKY              47 M 145
528  BARSON AARON V JR           48 M 108
528  HUNLEY MICHAEL W            47 M
528  HESTER JAMES CLITTON        47 M 089
528  SMITH LINDA DIANE           48 F 127
528  CHIDESTER MILTON WAYNE      51 M 061
528  MANN MICHAEL DOUGLAS        50 M 014
528  TAYLOR STEPHEN COPE         48 M 090
528  LAMOUREUX BLAIN C           49 M
529  BARFUS BRENT WAYNE          42 M 108
529  PETERSON GLEN LOREN         44 M 014
529  MURPHY DEE FRANKLIN         46 M 132
529  GUYMON MAUGHAN M            47 M 039
529  COPE PATRICIA               48 F 146
529  PREEDEDILOK KITIMA P         39 F
530  MARTIN LONNIE JOSEPH        47 M
530  BRIGGS BETTE MARLENE        48 F
532  REED FRANK E                28 M
538  JARRELL DANIEL CRAIG        47 M
539  BROWN LAWRENCE GUY          39 M
549  ANDRE RAMONA LOUISE E       34 M
550  MAJOR GARY LEE              45 M
552  PETERS DONALD CARL           47 M
555  SAADAT MOHAMAD H            50 M
557  EDWARDS RONALD DANE         47 M
561  KELSO THOMAS R              50 M
562  WEYERTS THOMAS MARTIN       M
<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Age</th>
<th>Gender</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>573</td>
<td>TURNQUIST GARY BRUCE</td>
<td>50</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>666</td>
<td>SIQUEIROS BRUCE WAYNE</td>
<td>49</td>
<td>M</td>
<td>282</td>
</tr>
<tr>
<td>777</td>
<td>WALKER CLIVE HANSEN</td>
<td>35</td>
<td>M</td>
<td>327</td>
</tr>
<tr>
<td>888</td>
<td>NASERI MOHSEN</td>
<td>50</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>999</td>
<td>SMITH DENNIS LYLE</td>
<td>47</td>
<td>M</td>
<td>127</td>
</tr>
</tbody>
</table>
VITA

Karl A. Fugal

Candidate for the Degree of

Master of Science

Thesis: A FORTRAN List Processor (FLIP)

Major Field: Applied Statistics

Biographical Information:

Personal Data: Born at American Fork, Utah, December 3, 1939, son of Bryan C. and Jennie Burch Fugal; married Carol Don Carpenter on August 26, 1962.

Education: Attended elementary school in Pleasant Grove, Utah; graduated from Pleasant Grove High School in 1958; received a Bachelor of Science degree from Utah State University with a major in mathematics and a minor in physics, in June, 1964; completed requirements for the Master of Science degree at Utah State University in 1970.